25

VLSI Architectures for Pattern Analysis and Image Database Management

KING-SUN FU, KAI HWANG, AND BENJAMIN W. WAH

Purdue University
West Lafayette, Indiana

25.1 INTRODUCTION

In this chapter VLSI computing structures are introduced for the analysis and management of imagery data. Information scientists have long recognized the fact that *one picture is worth a thousand words.* Machine intelligence would be greatly enhanced if a new generation of computers could be designed to process multidimensional imagery data above the string processing of alphanumerical information by present computers. Over the last decade, extensive research and development has enabled us to achieve the capabilities of pattern analysis and image understanding by computers. Practical applications of such computers include the analysis of biomedical images, the recognition of characters, fingerprints, and moving objects, remote sensing, industrial inspection, robotic vision, military intelligence, and data compression for communications [11,12,24].

Image analysis refers to the use of digital computers for *pattern recognition and image processsing* (PRIP). On-line imagery data need to be restored on disks and quickly retrieved for PRIP applications. A VLSI-based image analysis machine should integrate both pattern-analysis and image-database-management capabilities into a unified system design [1,10].

25.2 VLSI AND COMPUTER IMAGING

We are in the era of *very large scale integration* (VLSI). Integrated circuits have penetrated all areas of human civilization. IC wafer size has increased from 1 in. to 6 in. in 20 years. Bell Laboratories have produced some 8-in. wafers and 1 megabit

memory chips. Fujitsu in Japan has announced a new nonsilicon device, *high-electron mobility transistor* (HEMT), which has a 17-ps switching time, 30 times faster than the fastest silicon counterparts. The *very high speed integrated circuit* (VHSIC) project was challenged to produce 4-ps devices with some success. In the research community, *wafer-scale integration* (WSI) has been vigorously considered in implementing algorithmically specialized computer structures.

Advances in VLSI technology have triggered the thought of implementing many signal/image processing algorithms directly on specialized hardware chips. To promote image understanding, a back-end image database machine will be highly desirable in future computers. The new concept of data-driven computations can be adopted for artificial intelligence applications. New concepts on VLSI computing architectures and asynchronous data-flow multiprocessors should be seriously explored for potential use in image processing and pattern recognition. Extending control flow computers from two-dimensional arrays to three-dimensional pyramid structures is also a viable approach. Multiple-pipeline computers are also good candidates for parallel image processing if task scheduling problems can be efficiently solved.

The principal deficiency of today's computers lies in their input/output mechanisms. Computers still cannot communicate efficiently with human beings in natural forms, such as spoken or written languages, pictures or images, documents, and

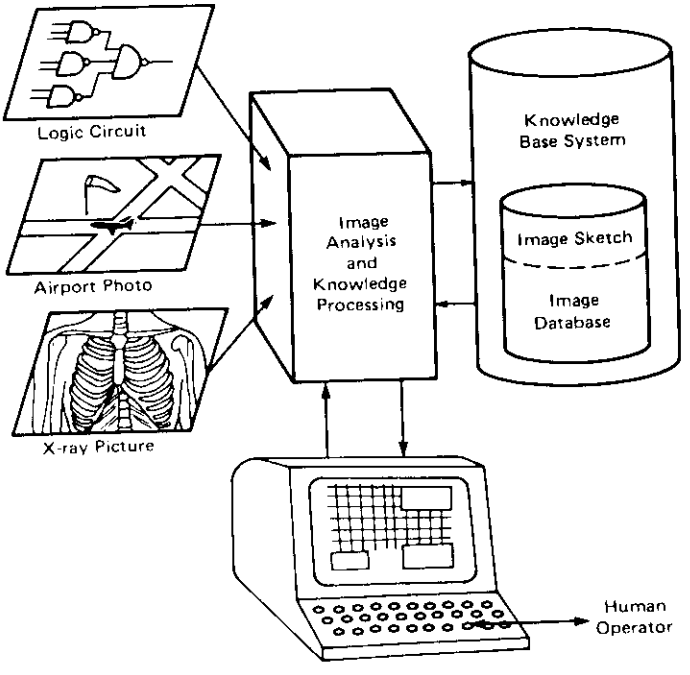


Figure 25.1 Image analysis/retrieval functions.

illustrations. Existing computers are far from satisfactory in their slowness in I/O and lack of speech, vision, translation, and real-time responses. To develop a "human-oriented" interactive computer will require, first, upgrading their capability to understand "natural" information representations and to respond to them intelligently and perhaps more reliably than human beings. Natural language and speech processing are beyond the scope of this presentation. We focus below on developing intelligent computers with image analysis/retrieval functions.

To establish these functions, one needs to develop subsystems for *input*, *output*, and *analysis* of imagery data retrieved from a large *image database system*. Pictorial functions of such an intelligent image analysis computer are conceptually illustrated in Figure 25.1. The input forms may be line drawings or gray-scaled images such as logic circuit diagrams, chest x-ray pictures, and airphoto images. The corresponding outputs may be the classification and/or interpretation of the input data, such as the layout of a VLSI circuit design, a precise description of the abnormality in the lung area, or a combat map generated in real time. The image database machine may be a part of a large-knowledge-base system. Many image-analysis functions need to be built into the middle processing section for image enhancement and segmentation, feature extraction, pattern classification, structural analysis, image description, and interpretation, as listed in Table 25.1. Some of these functions can be implemented directly by VLSI hardware and some by special software packages. Advances in image I/O devices, pictorial query processing, and image database management techniques are demanded to construct an integrated image analysis/retrieval system.

TABLE 25.1 CANDIDATE IMAGE ALGORITHMS FOR VLSI

Image processing	Enhancement, filtering, thining, edge detection, segmentation, registration, restoration, clustering, texture analysis, convolution, Fourier analysis, etc.
Pattern recognition	Feature extraction, template matching, statistical classification, graph algorithms, syntax analysis, change detection, language recognition, scene analysis and synthesis, etc.
Image query processing	Query decomposition, query optimization, attribute manipulation, picture reconstruction, search sorting algorithms, query-by-picture-example implementation, etc.
Image database processing	Relational operators (JOIN, UNION, INTERSECTION, PROJECTION, COMPLEMENT), image-sketch-relation conversion, similarity retrieval, data structures, priority queues, dynamic programming, spatial operators, etc.

25.3 VLSI IMAGE PROCESSORS

Recently, many attempts have been made in developing VLSI devices for signal/image processing and pattern analysis. The statistical approaches to PRIP often involve large-scale matrix computations. The structural approaches require the performance of syntax analysis and parsing operations. We present below example designs of pipelined VLSI architecture, for statistical feature extraction and classification. The pipeline stages are constructed with modular arithmetic devices as functionally specified in Figure 25.2. These VLSI arithmetic modules will be used

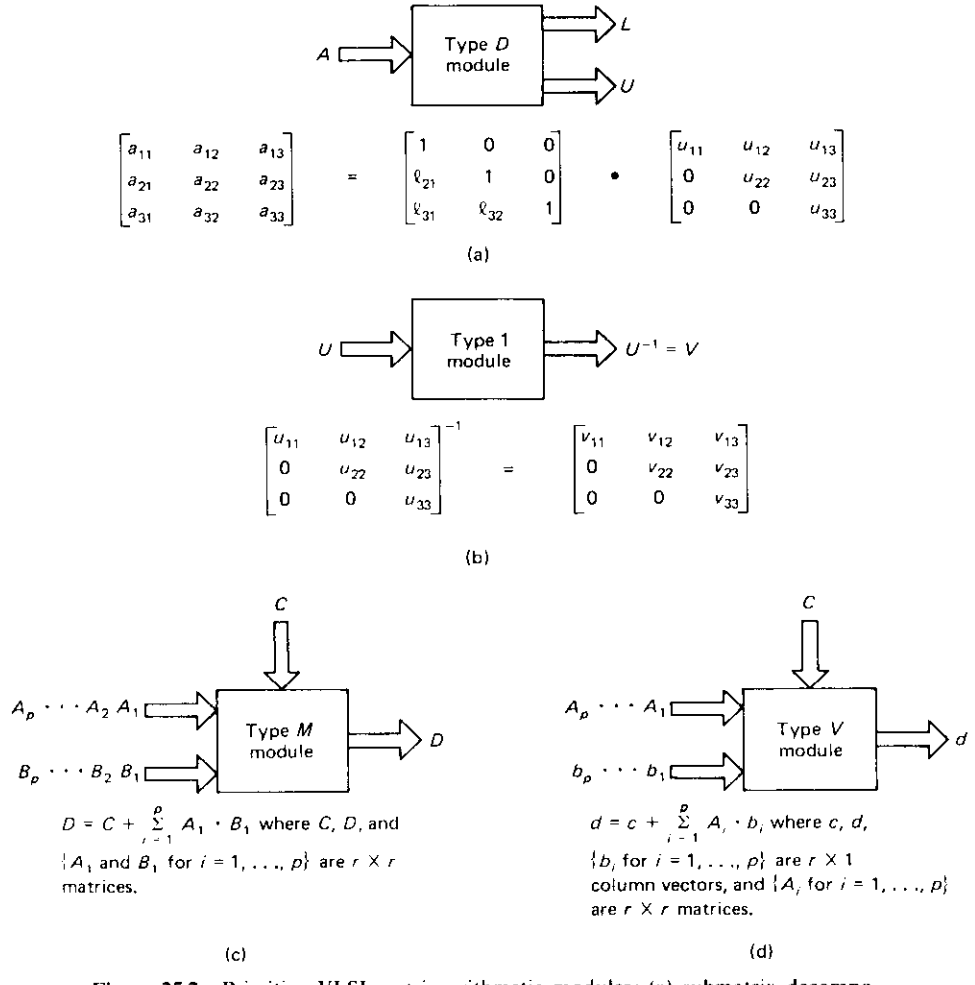
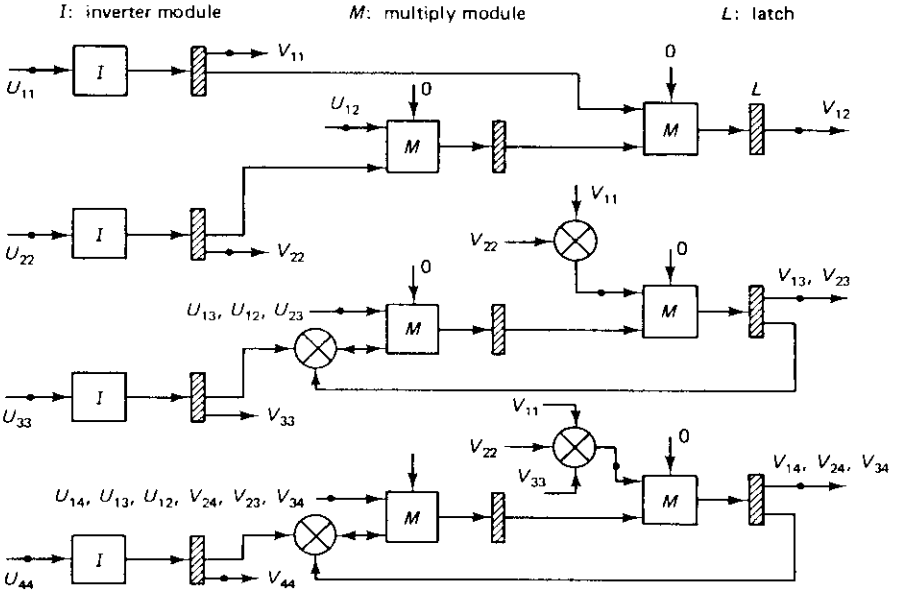


Figure 25.2 Primitive VLSI matrix arithmetic modules: (a) submatrix decomposition module; (b) submatrix inverter; (c) matrix multiplier; (d) matrix-vector multiplier.

iteratively in submatrix computations. This pipeline architecture is based on the "partitioned" matrix algorithms developed in [9] for *L-U decomposition, matrix multiplication, inversion of triangular matrices,* and *solving triangular systems of equations.*

Figure 25.3 shows the pipelined structure for implementing the "partitioned" *matrix inversion* algorithm being outlined in four steps. Each VLSI module performs an $r \times r$ submatrix computation where n is the order of the input matrix U. In practice , $n = kr$ and $n \gg r$ are assumed. The case of $k = n/r = 4$ is shown in Figure



Note: All U_{ij}, V_{ij} are $m \times m$ submatrices

$$U^{-1} = \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{23} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{41} \end{bmatrix}^{-1} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & V_{14} \\ 0 & V_{22} & V_{23} & V_{21} \\ 0 & 0 & V_{33} & V_{34} \\ 0 & 0 & 0 & V_{44} \end{bmatrix} = V$$

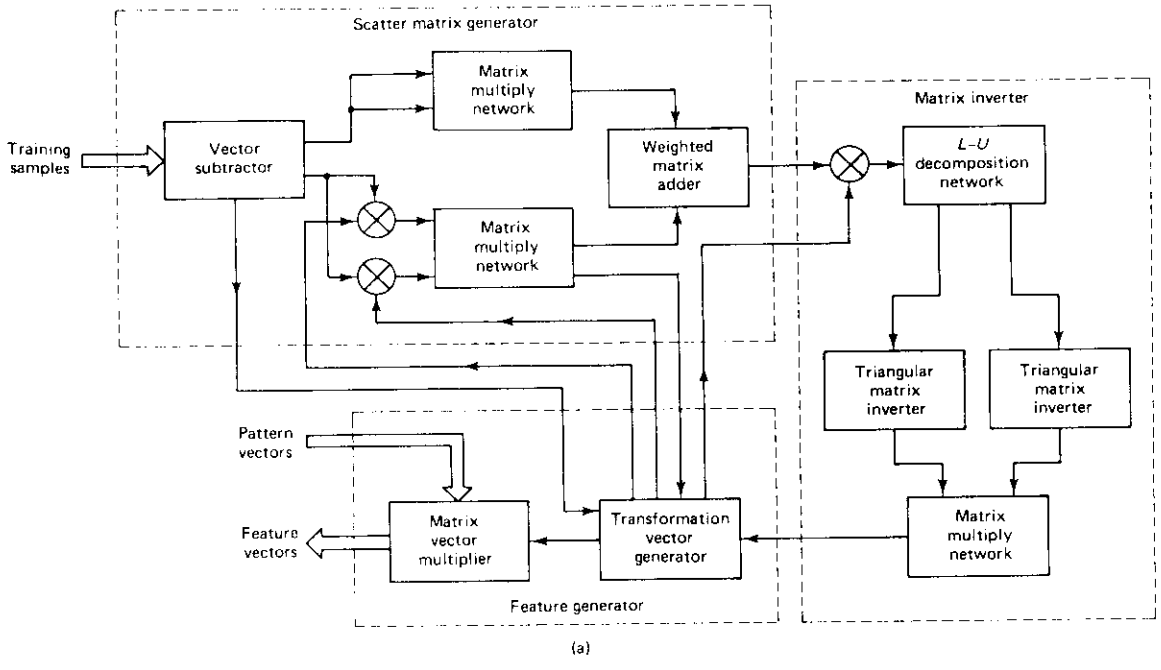
Step 1: $V_{11} = U_{11}^{-1}$; $V_{22} = U_{22}^{-1}$; $V_{33} = U_{33}^{-1}$; $V_{44} = U_{44}^{-1}$ (I modules)

Step 2: $V_{12} = -V_{11} \cdot (U_{12} \cdot V_{22})$; $V_{23} = -V_{22} \cdot (U_{23} \cdot V_{33})$; $V_{34} = -V_{33} \cdot (U_{34} \cdot V_{44})$ (M modules)

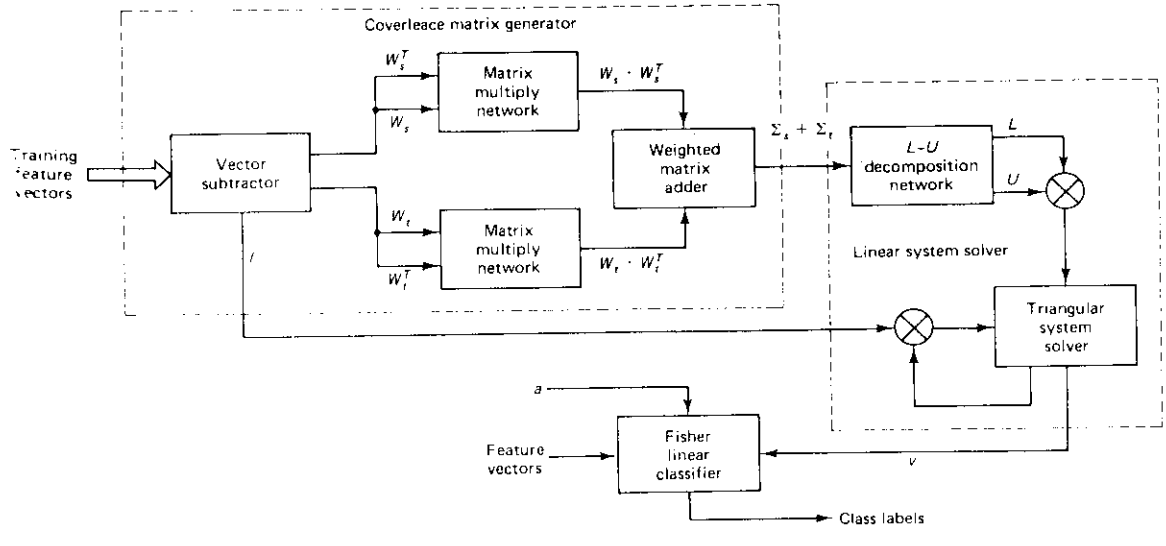
Step 3: $V_{13} = -V_{11} \cdot (U_{12} \cdot V_{23} + U_{13} \cdot V_{33})$
$V_{24} = -V_{22} \cdot (U_{23} \cdot V_{34} + U_{24} \cdot V_{44})$ (M modules)

Step 4: $V_{14} = -V_{11} \cdot (U_{12} \cdot V_{24} + U_{13} \cdot V_{31} + U_{14} \cdot V_{41})$ (M modules)

Figure 25.3 VLSI matrix inversion pipeline based on Hwang/Cheng's partitioned algorithm.



(a)



(b)

Figure 25.4 VLSI architectures for pattern analysis/recognition: (a) VLSI feature extractor; (b) VLSI pattern classifier.

25.3. For large k, this pipeline requires $O(k) = O(n/r)$ VLSI modules to implement. The total time delay to generate $V = U^{-1}$ will be $O(n^2/r)$. Similar arithmetic pipelines can be constructed for *matrix multiply*, *L-U decomposition*, and *solving triangular systems*. Details can be found in [8].

VLSI feature extraction. Figure 25.4(a) shows the functional design of a VLSI feature extractor. This extractor is constructed with three subsystems: *scatter matrix generator*, *matrix inverter*, and *feature generator*, as shown by dashed-line boxes. The vector subtractor is implemented with modified V modules for generating the sample offset matrices and the mean difference. Two matrix multiply networks are used to perform orthogonal matrix multiplications. Each network contains n/r M modules. The weighted matrix adder can be implemented by n/r M modules with some special constant inputs. The inversion of the scatter matrix is done by employing an L-U decomposition network, two triangular matrix inverters, and one multiply network to yield the computation $A^{-1} = (L \cdot U)^{-1} = U^{-1} \cdot L^{-1}$. The feature generator can be implemented by V modules with modified constant inputs. Finally, the matrix-vector multiplier is also implemented with V modules.

VLSI pattern classification. The functional design of a VLSI pattern classifier is sketched in Figure 25.4(b). The schematic design of the *covariance matrix generator* is similar to the scatter matrix generator in Figure 25.4(a). The *linear system solver* is composed of an L-U decomposition network and a triangular system solver. This matrix solver is needed to triangularize a dense system. The Fisher classifier is implemented by some combinational logic circuits and modified V modules.

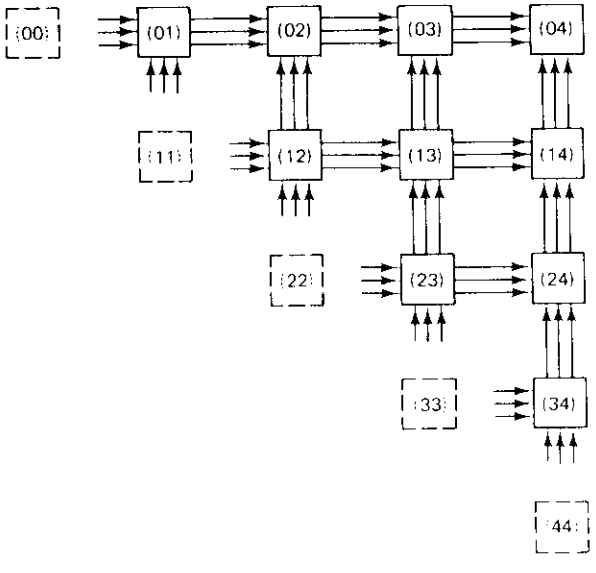


Figure 25.5 VLSI architecture for fast recognition of context-free languages (the case of $n = 4$ is shown). Solid square: cell; dashed square: boundary location

Context-free language recognition. In syntactic image analysis, an image pattern is often represented by a string in a context-free language. Recognition of an image pattern is accomplished through a parsing of the string with respect to a given pattern grammar [14]. A VLSI systolic array for high-speed recognition of context-free languages is shown in Figure 25.5. The recognition process is based on the Cocke–Kasami–Younger algorithm. This pipelined triangular array, constructed of $n(n+1)/2$ processing cells, can be applied in syntactic pattern recognition. Each cell has two unidirectional data channels and one control line along each direction. Data appear as strings of symbols flowing through the recognition matrix from left to right and bottom to top as shown in Figure 25.5. This two-dimensional array can recognize any input string of length n in $2n$ time units. This context-free language recognizer and its extension to recognize finite-state languages are described in more detail in [3]. A VLSI architecture for high-speed recognition of context-free languages using Earley's algorithm has recently been proposed [21].

VLSI seismic classification. A special-purpose VLSI processor is presented below for fast classification of seismic waveforms [20]. This special-purpose processor which contains three systolic arrays can be attached to a host computer.

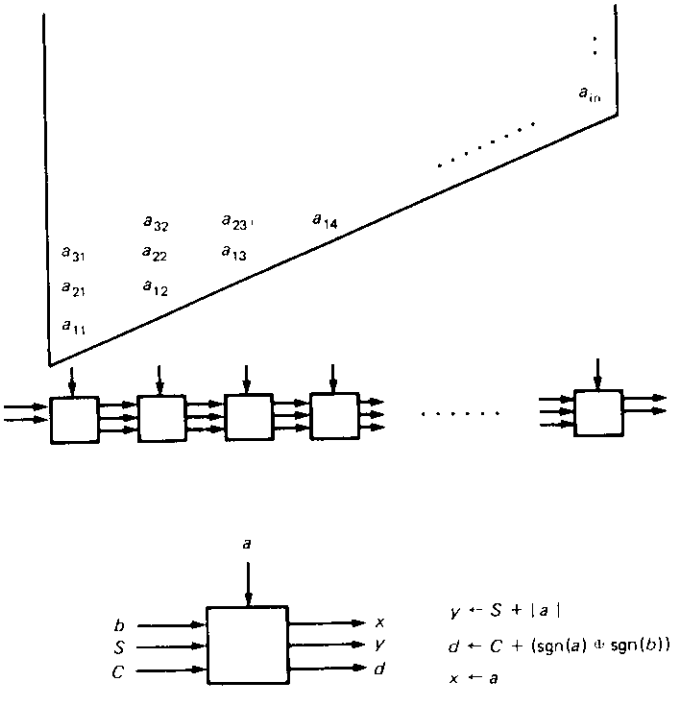


Figure 25.6 Processor array, data movement and operations of each processor for feature extraction.

Each systolic array has time complexity $O(1)$ provided that input data can be properly supplied.

The systolic array for feature extraction contains linearly connected processing elements, as shown in Figure 25.6. The input data, which are the digitized and quantized seismic waveform coded in binary form, are stored in separate memory modules in a skewed format. Two features, zero-crossing count and sum of absolute magnitudes, are computed. Zero crossing is detected by checking the signs of every two consecutive points. An exclusive-OR circuit is used for the detection of sign change. All the n processing elements (PEs) compute the two features simultaneously and pass the partial results to the next PEs.

For primitive recognition, we compute the distance between an unknown feature vector and each reference vector, for example, mean vector, of each cluster (primitive), and then assign the unknown feature vector to the cluster of the minimum distance. This procedure can be divided into two steps: compute the distances between the unknown feature vector and the reference vectors, and then select the

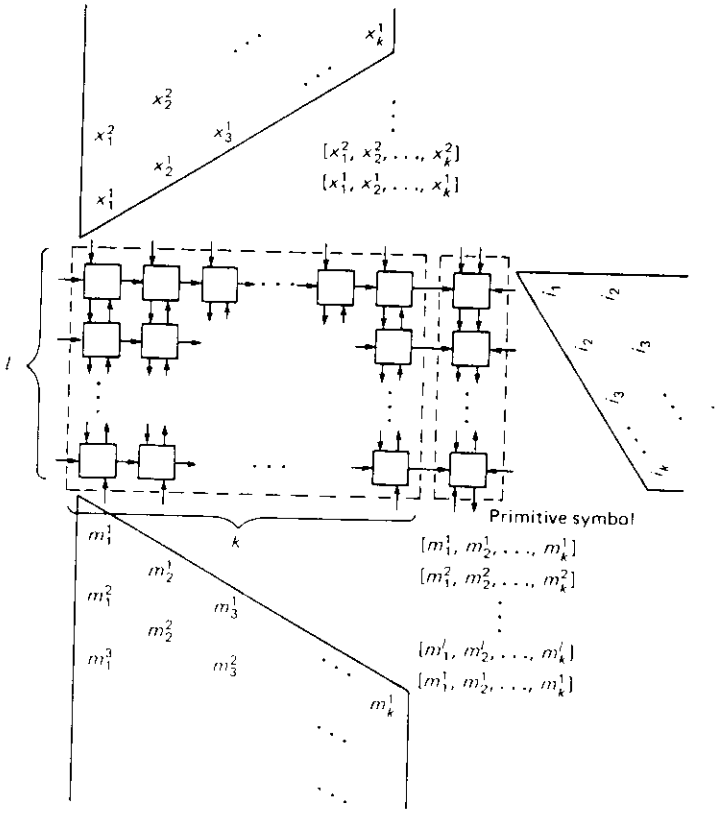


Figure 25.7 Processor arrays and data movement for primitive recognition.

one associated with the smallest distance. We use a processor array which contains *compute* processors for distance computation, and a processor array which contains *compare* processors for distance comparison. Suppose that there are l primitives; each primitive i has a reference feature vector $[m_1^i, m_2^i, \ldots, m_k^i]$, where k is the total number of features. A processor array of l by k which performs the distance computation is shown in Figure 25.7. The reference vectors of the primitives enter from the bottom and move up while the unknown feature vectors enter from the top and move down.

It is well known that the Levenshtein distance between two strings can be computed by a dynamic programming procedure. Therefore, it can be implemented by parallel processing on VLSI architectures. For Levenshtein distance, each insertion, deletion, and substitution is counted as one error transformation. We have developed a processor array for this string-matching computation in Figure 25.8. The proposed string matcher can be used for any problem where the Levenshtein distance computation is required. It can be used for string matching in our seismic recognition, for character string matching in information retrieval or for pattern

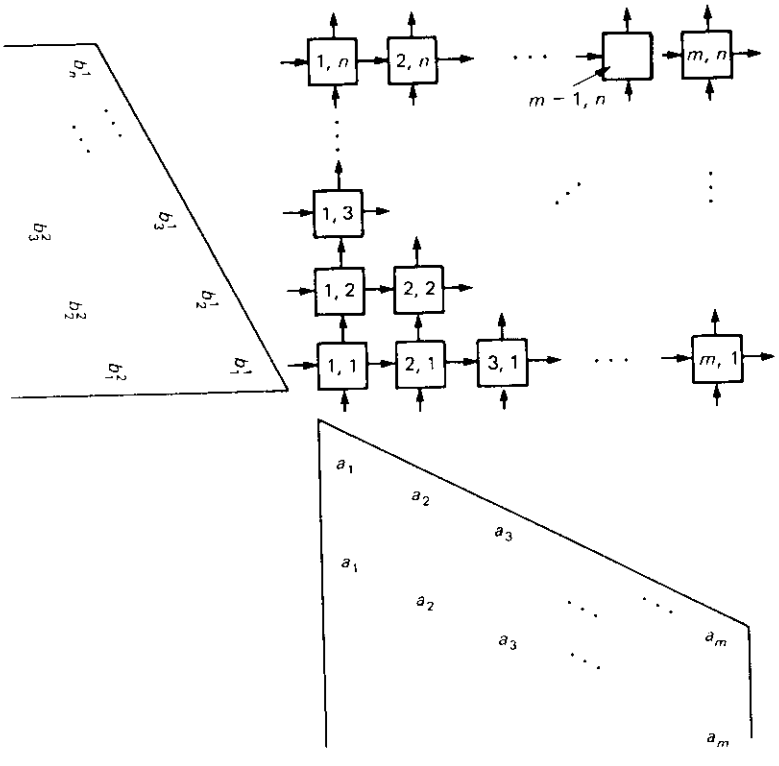


Figure 25.8 Processor array and data movement for computing Levenshtein distance.

matching in shape analysis if the object can be represented by a string, for example, using chain codes. The primitive recognizer can also be applied to any minimum-distance recognition problem and vector pattern matching.

Simulations have been performed for the three systolic arrays: feature extraction array, primitive recognition array, and string matching array. The design of the systolic arrays has been shown to be correct and the operations are as expected. Details of the systolic arrays and simulation results above were reported in Liu and Fu [20].

25.4 IMAGE DATABASE MACHINES

An image database system provides a large collection of structured imagery data (digitized pictures) for easy access by a large number of users. It provides both high-level quay support and low-level image access. Most image database systems are implemented with specially developed software packages upon dedicated pattern-analysis systems. It is highly desirable to develop a dedicated back-end database machine for image database management. So far, several hardware attempts were suggested [4,16,22]. But none of them has been actually implemented for image database management.

Image database management functions and peripheral supports are depicted in Figure 25.9. First, we need faster and intelligent image input devices. The image features and structures (shape, texture, and spatial relationships) extracted by the host image processor should be converted into symbolic image sketches stored in a *logical* image database. For those unconverted raw images, the system must convert them into efficient codes stored in the *physical* image database. Flexible image manipulation and retrieval functions must be established using high-level image manipulation languages and image description languages. The logical database is

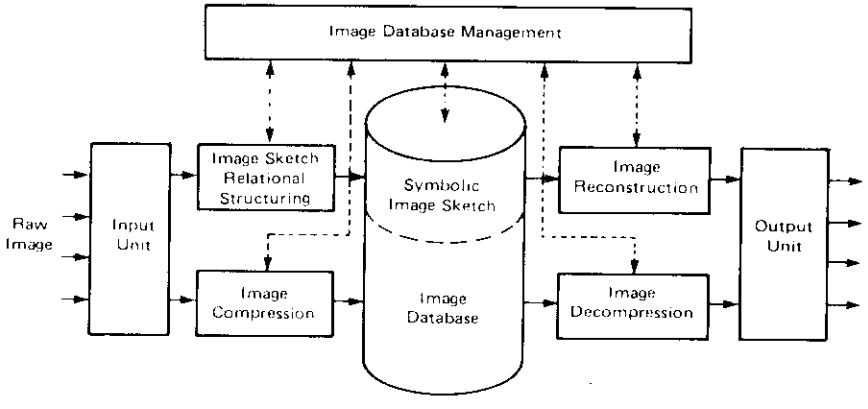


Figure 25.9 Image database management functions.

used for image reconstruction from relational sketches. The compressed raw images must be decompressed for high-resolution console display. The output unit is responsible for extracting results to be sent to the host computer. The image database management functions above should be supported with specially designed hardware units that constitute an image database machine.

A database machine for image processing can be identified to have the following functional features. High-level database functions such as selection, projection, and join are implemented. These operations are useful for manipulating the image database. On the other hand, low-level image processing operations such as histogramming and edge detection are also implemented. An image database machine is, therefore, a conventional database machine enhanced with low-level image-processing hardware.

A general assumption about VLSI chips is that they are inexpensive. For complex operations, this is not really true, due to the fact that external control, timing, memory, and software must be provided. Furthermore, as the types of VLSI chips increase and the degree of replication is large, the system becomes expensive. A solution to this problem is to use a resource-sharing interconnection network so that a pool of common resources can be used. This concept is illustrated in Figure 25.10. VLSI chips are distributed into each storage module. They can be used to real-time off-the-track processing. A pool of common resources are also shared

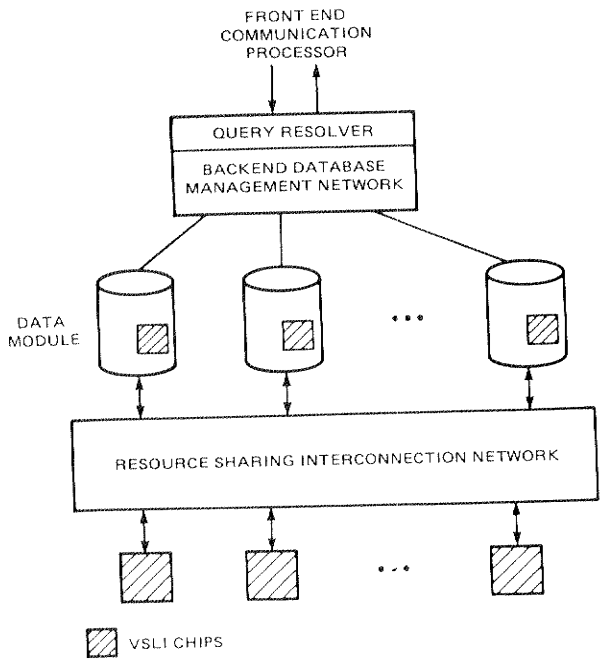


Figure 25.10 Conceptual view of an image database machine.

among the storage modules. The resource-sharing interconnection network connects these resources to the storage medium. The shared database operators will be used for data filtering, projection, join, or other operations if a relational image database is established. Some VLSI database operators are listed in Table 25.1.

5.5 DISTRIBUTED SCHEDULING OF RESOURCES

In general, an interconnection network routes requests from a set of source points to a set of destination points (they may coincide with each other). In a *resource-sharing mode*, the destination points are identical (or sets of identical) resources such as special-purpose VLSI chips for which requests or tasks can be delegated. In this respect, jobs initiated at source processors can be sent to any one of the free resources of a given type at the destination. This is the important point that differentiates resource sharing from address mapping.

Since the system operates continuously, requests from source processors can be initiated at random times. At any time, a set of processors may be making requests and a set of resources are free. It is the function of a scheduler to route the requests in order to connect the maximum number of resources to the processors, that is, to have the maximum resource utilization.

The earliest study of networks for resource sharing has been realized with centralized control. A unibus is used in a time-shared fashion for connecting peripheral I/O devices to the CPU. Multiple time-shared buses have been used in the PLURIBUS minicomputer multiprocessor. A cross-bar switch has been used in C.mmp, although the network is mostly used in the address mapping mode. The single- or multiple-bus approach is a source of bottleneck, and is the least expensive design. The cross-bar switch is the most expensive network but has the least degree of blocking. A compromise is to use a less expensive network than the cross-bar switch and which has less blocking probability than single-bus systems. This has been studied with respect to the Banyan network. A tree network is proposed to aid the scheduler in choosing a resource to allocate. The tree network has a delay of $O(\log_2 n)$ in selecting a free resource (n is the total number of resources).

The scheduling algorithms studied earlier are centralized and use address mapping interconnection networks. For mapping n requesting processors to n resources, the scheduling algorithm has a worst-case complexity of $O(n \log_2 n)$. This complexity depends on the number of requesting processors. This is practical when n is small or when requests are not very frequent. A solution that avoids the sequential scheduling of requests is to allow requests to be sent without any destination tags, and it is the responsibility of the network to route the maximum number of requests to the free resources. In this way, the scheduling intelligence is distributed in the interconnection network. This approach permits multiple requests to be routed simultaneously. We termed this network a *resource-sharing interconnection network*.

The Omega and generalized cube networks belong to a class of networks with

the property that the delay from a source to any reachable destination is proportional to the logarithm of the number of source points. The basic element in these networks is a two-input two-output four-function interchange box which allows a straight, exchange, upper broadcast, or lower broadcast connection. For a network connecting N inputs to N outputs (N is a power of 2), there are $\log_2 N$ stages and $(N/2) \log_2 N$ interchange boxes. The delay in the networks is, therefore, $O(\log_2 N)$. The $O(N \log_2 N)$ hardware complexity is much better than that of the cross-bar switches, $O(N^2)$.

Since the networks have nonzero blocking probability, some of the feasible mappings from sources to destinations do not lead to maximal resource allocation. A centralized scheduler has to examine all the different possible ordered mappings in order to allocate the maximum number of resources. Suppose that x processors are making requests and y resources are free. The scheduler has to try a maximum of $\binom{x}{y} y!$ (for $x \geq y$) or $\binom{y}{x} x!$ (for $y > x$) mappings in order to find the best one. Suboptimal heuristics can be used [19], but will only be practical when x and y are small.

On the other hand, a distributed scheduling algorithm allows all the requests to be scheduled in parallel. The resource scheduling overhead is, therefore, proportional to the delay time in the network, $O(\log_2 N)$, and independent of the number of requesting processors.

The distributed algorithm is implemented by distributing the routing intelligence into the interconnection network so that there is no centralized control. Each exchange box can resolve conflicts and route requests to the appropriate destinations. If a request is blocked, it will be sent back to the originating exchange box in the previous stage. Request routing is thus dynamic, and all the exchange boxes operate independently.

The distributed algorithm is illustrated in Figure 25.11 on an 8×8 Omega network. Suppose that resources R_0, R_1, R_4, and R_5 are available and status information is passed to the processors. The numbers on the output/input ports represent the status information received/sent. Assuming that P_0, P_3, P_4, and P_5 are requesting one resource each, the requests are sent simultaneously to the network after new status information arrives. In stage 0, no conflict is encountered. $B_{1,1}$ in stage 1 receives two requests. Since only one output terminal leads to free resources, the request originating from $B_{0,3}$ is rejected. This request, subsequently, finds another route via $B_{1,3}$ and $B_{2,2}$ to R_5. In this example, each request has to pass through 3.5 exchange boxes on the average before it finds a free resource. For clarity, status changes due to new requests are not indicated in the figure.

The resource-sharing network discussed here is a generalization of address-mapping interconnection networks with routing tags. An address-mapping network is a resource-sharing network connecting processors and multiple types of resources with one resource in each type. In a resource-sharing mode, multiple resources are allowed in each type. This resource-sharing network has a uniform structure suitable for VLSI implementation.

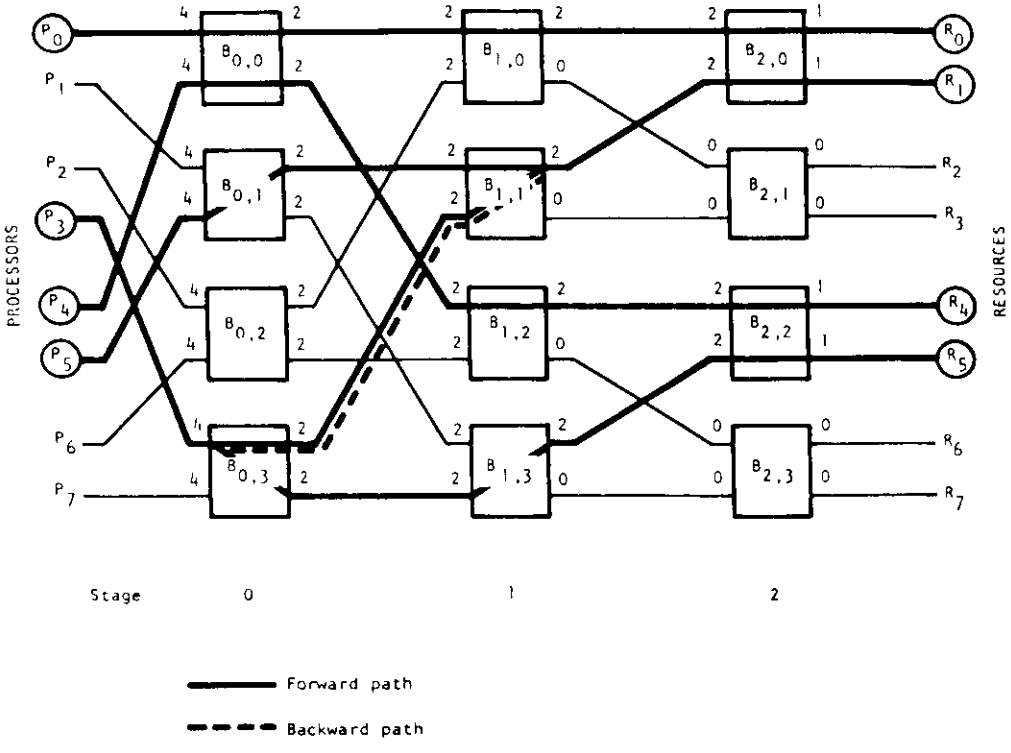


Figure 25.11 Example of Omega network with four requesting processors and four free resources (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 3.50 units).

25.6 CONCLUDING REMARKS

Feature extraction and pattern classification are initial candidates for possible VLSI implementation. The Foley–Sammon feature extraction method [5] and the Fisher linear classifier have been proposed for VLSI implementation [8]. Other methods, such as the eigenvector approaches to feature selection and Bayes's quadratic discriminant functions, should be also realizable with VLSI hardware. It is highly desirable to develop VLSI computing structures for smoothing, image registration, edge detection, image segmentation, texture analysis, multistage feature selection, syntactic pattern recognition, pictorial query processing, image database management, and so on. The potential merit lies not only in speed gains, but also in reliability and cost-effectiveness.

Image analysis and image database management are two functions that cannot be separated in an efficient pictorial information system. The integrated system approach is supported by the merging VLSI technology and by various parallel

processing techniques. Cost-effectiveness is the key issue in developing special-purpose machines for image processing, recognition, and database management. Toward the eventual VLSI realization of an integrated image analysis/retrieval computer, we suggest below a number of important research projects.

1. Develop systematic design methodology for mapping PRIP algorithms into VLSI hardware architectures [3,6–9,15,20,21].
2. Develop VLSI devices for image description, image manipulation, and pictorial query processing [2,11,13,17].
3. Develop back-end image database machines, including both image database structures and management policies [4,16,22].
4. Develop the resource arbitration networks for a multiprocessor system with a shared VLSI resource pool [1,18,19].
5. Investigate the possible use of the data flow concept in designing VLSI systems for PRIP and artificial intelligence computations [1,7,23].
6. Integrate VLSI architectures for image analysis with those for natural language and speech processing [10,23,24].

ACKNOWLEDGMENT

This work was supported by National Science Foundation Grant ECS 80-16580.

REFERENCES

[1] F. A. Briggs, K. S. Fu, K. Hwang, and B. W. Wah, "PUMPS Architecture for Pattern Analysis and Image Database Management," *IEEE Trans. Comput.*, Oct. 1982, pp. 969–982.

[2] S. K. Chang, J. Reuss, and B. H. McCormick, "Design Considerations of a Pictorial Database System," *Intl. J. Policy Anal. Inf. Syst.*, *1*(2):49–70 (Jan. 1978).

[3] K. H. Chu and K. S. Fu, "VLSI Architectures for High-Speed Recognition of General Context-Free Languages and Finite-State Languages," *Proc. 9th Intl. Symp. Comput. Arch.*, Austin, Tex., Apr. 1982, pp. 43–49.

[4] T. DeWitt, "DIRECT: A Multiprocessor Database Machine," *IEEE Trans. Comput.*, 1979, pp. 395–406.

[5] D. H. Foley and J. W. Sammon, "An Optimal Set of Discriminant Vectors," *IEEE Trans. Comput.*, Mar. 1975, pp. 281–289.

[6] M. J. Foster and H. T. Kung, "The Design of Special-Purpose VLSI Chips," *Comput. Mag.*, Jan. 1980, pp. 26–40.

[7] K. Hwang and F. A. Briggs, *Computer Architectures for Parallel Processing*, McGraw-Hill, New York (in press to appear).

[8] K. Hwang and S. P. Su, "VLSI Architectures for Feature Extraction and Pattern Classification," *J. Comput. Graphics Image Process.*, (accepted to appear in 1983).

[9] K. Hwang and Y. H. Cheng, "Partitioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Trans. Comput.*, Dec. 1982, pp. 1215–1224.

[10] K. Hwang and K. S. Fu, "Integrated Computer Architectures for Pattern Analysis and Image Database Management," *Computer*, Jan. 1983, pp. 51–61.

[11] M. Onoe, K. Preston, and A. Rosenfeld, eds., *Real-Time/Parallel Computing: Image Analysis*, Plenum Press, New York, 1981.

[12] K. Preston, Jr., and L. Uhr, eds., *Multicomputers and Image Processing*, Academic Press, New York, 1982.

[13] N. S. Chang and K. S. Fu, "Picture Query Languages for Pictorial Database Systems," *Computer*, *14*, Nov. 1981.

[14] K. S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[15] E. E. Swartzlander, "VLSI Architecture," in D. F. Barbe, ed., *Very Large Scale Integration (VLSI): Fundamentals and Applications*, Springer-Verlag, New York, 1980.

[16] B. W. Wah and S. B. Yao, "DIALOG—A Distributed Processor Organization for Database Machines," *AFIPS Conf. Proc.*, Vol. 49, 1980, NCC, pp. 243–253.

[17] M. Yamamura, N. Kamibayashi, and T. Ichikawa, "Organization of an Image Database Manipulation System," *Proc. Workshop Comput. Arch. for PAIDM*, Hot Springs, Va., 1981, pp. 236–241.

[18] D. W. L. Yen and A. V. Kulkami, "The ESL Systolic Processor for Signal and Image Processing," *Proc. Workshop Comput. Arch. for PAIDM*, Hot Springs, Va., 1981, pp. 265–272.

[19] B. W. Wah, "A Comparative Study of Resource Sharing on Multiprocessors," *IEEE Trans. Comput.*, Aug. 1984.

[20] H. H. Liu and K. S. Fu, "VLSI Systolic Processor for Fast Seismic Classification," *Proc. 1983 Intl. Symp. VLSI Tech. Syst. Appl.*, Taipei, Taiwan, Mar. 31, 1983.

[21] Y. T. Chiang and K. S. Fu, "A VLSI Architecture for Fast Context-Free Language Recognition (Earley's Algorithm)," *Proc. 3rd Intl. Conf. Distributed Comput. Syst.*, Oct. 1982.

[22] K. Yamaguchi and T. L. Kunii, "PICCOLO Logic for a Picture Database Computer and Its Implementation," *IEEE Trans. Comput.*, Oct. 1982, pp. 983–996.

[23] K. Hwang, "Computer Architectures for Image Processing," (Guest Editor's Introduction), *Computer*, Jan. 1983, pp. 10–13.

[24] K. S. Fu, ed., *Applications of Pattern Recognition*, CRC Press, Boca Raton, Fla., 1982.